Esercizio 1 (tema d'esame di Febbraio 2024)
Si vuole implementare un programma che legga da terminale un numero intero `N`, che rappresenta il numero di righe successive che il programma leggerà. Ciascuna di tali righe sarà composta da 8 caratteri (non sono previste situazioni di errore in tal senso). Il programma memorizza i caratteri letti in una matrice di caratteri `N`x8.

Successivamente, il programma legge una sequenza di lunghezza arbitraria di stringhe, ciascuna di lunghezza massima pari a 8 caratteri (anche qui non sono previste situazioni di errore).

In risposta a ciascuna stringa in ingresso, il programma stampa, oltre alla stringa stessa, seguita da ':', `1` se la stringa può essere trovata nella matrice come sottostringa di una delle righe, `2` se può essere trovata, ma leggendola al contrario, `3` se può essere trovata in verticale (dall'alto verso il basso) o `4` se può essere trovata in verticale, ma dal basso verso l'alto. Se vi sono più possibili risposte, viene data precedenza a quella numericamente più piccola.

Il programma stampa invece `0` se (e solo se) la stringa non può essere trovata.

Esempi:

stdin	**stdout**
4	assai:1
cerdsafi	sei:2
assaiesf	buoi:1
sofecrew	no:2
onbuoire	caso:3
assai sei buoi no caso zona nose	zona:0
	nose:4

Implementare la funzione `leggi_matrice`, che esegue la prima operazione, e la funzione `trova`, che determina, data la matrice letta `m`, il numero di righe `n` della matrice, e una stringa `s`, se la stringa `s` è presente nella matrice (restituendo un interno che segue le regole di cui sopra).

Codice fornito (e da non modificare):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define C 8

void leggi_matrice(char *m, int n);

int trova(char *m, int n, char *s);

int main(){

      int n;
      char w[C + 1];

      scanf("%d\n", &n);

      char *g = malloc(n * C * sizeof(char));
      if (!g) return 0;

      leggi_matrice(g, n);

      while (scanf("%s", w) == 1) {
```

```
            printf("%s:%d\n", w, trova(g,n,w));
        }

        free(g);
        return 0;
}
```

Esercizio 2 (tema d'esame di Febbraio 2024)
Si vuole implementare un programma che legga da terminale una stringa, che rappresenta il nome di un file. Il programma apre il file in questione, che contiene una lista della spesa, composta da un oggetto per linea. Ogni oggetto è definito dal proprio nome (una stringa lunga al più 31 caratteri), una quantità (numero intero) e un costo.

Il programma rappresenta la lista della spesa in memoria attraverso la struttura dati di tipo spesa_t riportata nel codice dato.

Implementare la funzione `leggi_lista` che, dato il nome del file contenente la lista, restituisce la corrisponde struttura dati di tipo `spesa_t` opportunamente popolata.
Implementare inoltre la funzione `totale`, che calcola il costo totale della spesa, e la funzione `mediano`, che calcola l'elemento di costo mediano (a questo scopo, si assume che le liste siano tutte composte da un numero dispari di elementi, oppure da nessun elemento).

Codice fornito (e da non modificare):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXSTRLEN 31

typedef struct {
	int n;
	char **oggetto;
	int *quantita;
	float *costo;
} spesa_t;

spesa_t leggi_lista(char *nomefile);

float totale(spesa_t *l);

char *mediano(spesa_t *l);

int main() {
	char nome[256];
	spesa_t s;
	scanf("%s", nome);
	s = leggi_lista(nome);
	printf("%.2f\n", totale(&s));
	printf("%s\n", mediano(&s));

	if (s.n > 0) {
		for (int i = 0; i < s.n; ++i) {
			free(s.oggetto[i]);
		}

		free(s.oggetto);
		free(s.quantita);
		free(s.costo);
	}

	return 0;
}
```

Esercizio 3 (tema d’esame di Febbraio 2024)
Il programma legge da stdin tre stringhe di massimo 32 caratteri e intende poi determinare se, usando tutti i caratteri delle prime due stringhe, è possibile costruire la terza mantenendo l'ordine relativo in cui i caratteri compaiono nelle prime due.

Nel caso in cui questa operazione sia possibile, il programma scrive su stdout il valore 1, altrimenti 0.

Si considerino le prime due stringhe come composte da insiemi disgiunti di caratteri.

stdin	**stdout**
`aa bb abba`	`1`

stdin	**stdout**
`aa bb aba`	`0`

stdin	**stdout**
`ab cd abda`	`0`

Il programma contiene quattro errori di sintassi che ne impediscono la corretta compilazione, ed alcuni errori logici che ne impediscono il corretto funzionamento. Trovare e correggere tali errori.

```
#include <stdio.h>
#include <string.h>
#define SLEN -32

int intreccio(char *s1, char *s2, char *s3){
        int j=0,k=0;
        for(int i=0; i<=strlen(s3); i++){
                if (s1[j]==s3[i]) j++
                elif (s2[k]==s3[i]) k++;
                else return 0;
        }
        return (strlen(s1)==k && strlen(s2)==j);
}

int main(){
        char str1[SLEN];
        char str2[SLEN];
        char str3[SLEN];
        scanf("%s %s %s", str3, str2, str1);
        printf("%d\n",intreccio(str1,str2,str3);
        return 0;
}
```

Esercizio 4 (tema d’esame di Febbraio 2023)
Implementare una funzione di prototipo `char *neg(char *s)` che riceve una stringa contenente caratteri '0' e '1', interpretata come un numero binario n (la cifra più significativa è nella posizione 0), e restituisce una nuova stringa che contiene il numero `-n` (opposto), espresso sullo stesso numero di bit.

Esercizio 5 (tema d’esame di Febbraio 2023)
Scrivere una funzione di prototipo:

```
int isogramma(char *s);
```

che determina se la stringa s è un isogramma, ovvero è composta da lettere che ricorrono tutte lo stesso numero di volte. Ad esempio, la parola "prosciugante" è un isogramma, precisamente il più lungo nella lingua italiana.

Le lettere minuscole e maiuscole vengono considerate identiche, quindi, ad esempio, "Anna" è un isogramma. La funzione restituisce la numerosità di ciascuna lettera nell'isogramma, oppure 0 se la parola non è un isogramma.

Esercizio 6.1 (tema d'esame di Febbraio 2023)

Data una matrice di interi con R righe e C colonne, scrivere un sottoprogramma che riorganizzi:

- gli elementi lungo le righe di indice **dispari**, come sequenze di valori che scandite da sinistra a destra esibiscono in **colonna [0] il massimo** valore nelle colonne 0...C-1, in **colonna [1] il minimo** valore nelle colone 1...C-1, in colonna [2] il massimo valore nelle colonne 2...C-1, in colonna [3] il minimo valore nelle colonne 3...C-1 e così via alternando massimi e minimi con lo stesso criterio fino a esaurimento dei valori sulla riga.

- gli elementi lungo le righe di indice pari, come sequenze di valori che scandite da sinistra a destra esibiscono in **colonna [0] il minimo** valore nelle colone 0...C-1, in **colonna [1] il massimo** valore nelle colonne 1...C-1, in colonna [2] il minimo valore nelle colonne 2...C-1, in colonna [3] il massimo valore nelle colonne 3...C-1 e così via alternando minimi e massimi con lo stesso criterio fino a esaurimento dei valori sulla riga.

Esempio

9	6	8	3	7
4	5	7	1	3
7	11	0	9	11

3	8	3	7	7
7	1	7	1	3
0	11	0	11	11

Suggerimento:
organizzare il codice prevedendo almeno due procedure aggiuntive per il riordino di una riga, che possano essere invocate come segue:

```
riorganizza_crescente(&M[i][0], colonne);       // i == valore pari
riorganizza_decrescente(&M[i][0], colonne);     // i == valore dispari
```

Esercizio 6.1 (tema d'esame alternativo di Febbraio 2023)
Data una matrice di interi con R righe e C colonne, scrivere un sottoprogramma che riorganizzi:

- Per le righe di indice dispari:
  - Per le colonne di indice pari: inserire i valori in ordine decrescente partendo dal massimo.
  - Per le colonne di indice dispari: inserire i valori in ordine crescente partendo dal minimo.
- Per le righe di indice pari:
  - Per le colonne di indice pari: inserire i valori in ordine crescente partendo dal minimo.
  - Per le colonne di indice dispari: inserire i valori in ordine decrescente partendo dal massimo.

Esempio

9	6	8	3	7
4	5	7	1	3
7	11	0	9	11

3	9	6	8	7
7	1	5	3	4
0	11	7	11	9

Suggerimento:
organizzare il codice prevedendo almeno due procedure aggiuntive per il riordino di una riga, che possano essere invocate come segue:

```
riorganizza_crescente(&M[i][0], colonne);       // i == valore pari
riorganizza_decrescente(&M[i][0], colonne);     // i == valore dispari
```